

MORSL: Minimal-Overhead Rank-Based Predictor with Summation-Free Correction and Lazy Access

Toru Koizumi*, Masanari Mizuno*, Soma Nishida*, Kanata Abe†, Tomoaki Tsumura*, Ryota Shioya†

*Nagoya Institute of Technology, Nagoya, Japan †The University of Tokyo, Tokyo, Japan

koizumi@nitech.ac.jp, m_mizuno@matlab.nitech.ac.jp, nishida@matlab.nitech.ac.jp,

abe@rsg.ci.i.u-tokyo.ac.jp, tsumura@acm.org, shioya@ci.i.u-tokyo.ac.jp

Abstract—Academic research on high-accuracy branch predictors has mainly focused on improving prediction accuracy under a fixed storage budget. However, for branch predictors in practical processors, implementation costs that affect energy consumption and latency are as important as storage capacity. This paper proposes MORSL, a cost-effective high-accuracy branch predictor. MORSL is a high-throughput predictor based on an ahead-pipelined rank-based TAGE predictor and introduces three mechanisms to improve energy efficiency. First, MORSL introduces narrow-word tagged tables, which reduce access energy by limiting each tagged-table entry to a single tag and compensate for the accuracy loss caused by low associativity through the choice of history-length sets and replacement policy. Second, we propose a tagged corrector as an auxiliary predictor that replaces summation-based auxiliary components and exploits auxiliary correlations without an adder datapath. Third, MORSL uses an allocation-guided access filter to suppress long-history tagged-bank accesses for easily predictable branches. We implement MORSL for HRCOM and evaluate it using 168 training traces from CBP-NG. The 37.5 KiB MORSL achieves a prediction accuracy of 4.803 MPKI and a VFS of 0.9935.

Index Terms—branch prediction, energy efficiency, prediction throughput, predictor accuracy

I. INTRODUCTION

Branch prediction research has long used storage capacity as the primary proxy for implementation cost [1]–[5]. Under this metric, the main objective has been to reduce MPKI within the same storage budget, and the central design problem has been to represent useful correlations efficiently with limited numbers of entries and counters. Along this line of research, longest-match-based tagged prediction and summation-based prediction, which integrates correlations from diverse features through summation, have evolved. Recent academic high-accuracy branch predictors, such as TAGE with statistical corrector (TAGE-SC) [19], [20] and multiperspective perceptron with TAGE [12], combine these two approaches and achieve high prediction accuracy under a fixed storage budget. These two approaches also form the core of high-accuracy branch predictors in practical processors, where they have been adopted as TAGE-like predictors and perceptron-based predictors [7], [9].

However, for branch predictors in practical processors, prediction accuracy under a fixed storage budget is not the only concern. Implementation factors that affect energy consumption and latency are also important. Even predictors with the same storage capacity can differ substantially in energy

consumption and latency depending on the number of SRAM arrays, word width, associativity, and banking organization. In addition, the cost of combinational logic, such as tag comparisons, longest-match selection, and adder datapaths, is also non-negligible. From this perspective, Seznec recently presented a realistic TAGE-SC organization that reduces the number of TAGE-SC tables and prediction latency by sharing physical tagged tables and simplifying the statistical corrector (SC) [21].

Despite these advances, three challenges remain in building energy-efficient high-accuracy branch predictors. First, reducing only the number of SRAM arrays is insufficient to reduce SRAM access energy. Even with fewer SRAM arrays, wide word widths and high associativity can still lead to high access energy. Second, the energy efficiency of summation-based auxiliary components must be reconsidered. Because a datapath that adds outputs from multiple tables increases energy consumption and latency, its accuracy benefit must justify its cost. Third, tagged-table accesses for easily predictable branches are often wasteful. Many branches can be predicted accurately using directional bias or short-history correlations. For such branches, the energy spent on TAGE tagged-table accesses often does not justify the resulting accuracy improvement. However, safely identifying such opportunities is not straightforward: before bypassing long-history tagged banks, the predictor must infer, without reading them, that they are unlikely to contain a useful matching entry.

To address these challenges, we propose MORSL, a cost-effective high-accuracy branch predictor. MORSL is a high-throughput predictor based on an ahead-pipelined rank-based TAGE predictor and introduces the following three mechanisms to improve energy efficiency while building on insights from TAGE-SC-like predictors.

- 1) **Narrow-word tagged tables:** We introduce a tagged-table organization that prioritizes narrow word widths, for example, by limiting each tagged-table entry to a single tag. MORSL compensates for the accuracy loss caused by this low-associativity organization through the choice of history-length sets and the replacement algorithm.
- 2) **Tagged correction:** We propose a tagged corrector (TC) as an auxiliary predictor that replaces a summation-based SC. The TC exploits auxiliary correlations without a datapath that adds outputs from multiple tables and

Allocation-guided access filtering: MORSL records past allocations of long-history TAGE entries and uses this information to bypass long-history tagged-bank accesses when they are unlikely to be useful. The allocation-guided access filter reduces dynamic energy consumption with little loss in prediction accuracy.

II. RELATED WORK

TAGE is one of the most accurate global-history-based branch predictors [23]. TAGE uses tagged tables with multiple history lengths and generally relies on the prediction from the longest matching history. A representative optimization for TAGE is bank interleaving [18]. Bank interleaving shares multiple banks among groups of history lengths, thereby mitigating imbalance in useful history lengths across programs and improving bank utilization. Seznec recently showed that large-scale interleaving is unnecessary and that 2:2 interleaving is a practical option [21]. However, the word width and associativity of each bank remain separate design choices.

To achieve high fetch throughput, block-wise prediction, in which a branch predictor predicts multiple branch instructions at once, is effective. Block-wise prediction generates predictions for multiple branch instructions in a fetch block or in a predictor-defined block. In this scheme, the branch direction history at the beginning of a block can differ from that at a branch in the middle of the block, making conventional direction histories difficult to use directly. Some history representations, such as path history, lghist [22] and target hash [10], can mitigate this problem.

Ahead pipelining has long been used to hide the latency of multi-cycle branch predictors. Because ahead pipelining starts table accesses using PCs or history information before the actual prediction target is known, the predictor needs to compensate for the accuracy loss caused by missing part of the history up to the target [8], [21], [22].

TAGE-SC improves prediction accuracy by combining TAGE with a statistical corrector (SC) [19], [20]. The SC reads multiple counter tables indexed by multiple features and integrates their outputs through summation. The multiperspective perceptron is another representative summation-based predictor that integrates correlations from diverse features through summation [11], [12]. These techniques can compensate for statistical biases and complex correlations that TAGE does not handle well, but they require a datapath that adds multiple outputs.

MORSL combines an ahead-pipelined TAGE predictor for rank-based block prediction with a tagged corrector (TC) as an auxiliary predictor and an allocation-guided access filter. Figure 1 (a) shows the organization of MORSL, and Figure 2 shows its prediction timing. MORSL generates a prediction vector for up to four conditional branches in a block using the base predictor and tagged tables. MORSL uses TAGE as the main predictor, while the TC overrides the TAGE prediction when appropriate to capture statistical biases and special history correlations that TAGE does not handle well. This organization avoids an adder datapath by implementing the auxiliary predictor as a tagged predictor rather than as a summation-based component. When the predictor determines that long-history tagged tables are unlikely to be needed, the



allocation-guided access filter operates the TAGE component of MORSL in mini-TAGE mode, which reads only short-history tagged tables and reduces the dynamic energy consumption of tagged-table accesses. With these mechanisms, MORSL improves the balance among prediction throughput, prediction accuracy, and energy efficiency while exploiting long-history correlations in TAGE.

IV. DETAILED PREDICTOR OPERATION

A. TAGE Organization

1) *Base and Tagged Predictor*: We adopt a rank-based TAGE predictor with eight different history lengths as the main predictor, as shown in Figure 1 (b). The predictor treats the instruction sequence up to the first taken branch, up to four conditional branches, or up to 32 instructions, possibly crossing the 128-byte boundary, as one block and performs rank-based prediction for the conditional branches in the block. The base predictor is a bimodal table indexed by the block-start PC and generates a prediction vector for each rank. The tagged predictor consists of tagged tables with eight history lengths. Each entry in a bank consists of a tag, a prediction counter, and a usefulness bit. The eight physical banks use 2:2 interleaving across bank pairs.

2) *History Lengths and Path History*: The history-length set places short histories densely and spreads long histories more widely. This organization follows recent work that uses a second-order arithmetic progression [14]. In particular, we expected the dense placement of short histories to act as pseudo-skewed associativity in a low-associativity organization where each entry has only one tag. In addition, using a relatively long shortest history length was also important, presumably because it compensates for the lack of information caused by ahead access.

MORSL uses path history updated by rules similar to those of lghist. This path history incorporates a 6-bit value derived from the lower address bits of the taken branch instruction itself using XOR. We do not use a target-only scheme because it makes branches in a block with the same target impossible to distinguish, thereby degrading prediction accuracy.

3) *Prediction Operation*: The indices and bank mappings of the tagged predictor are generated one block ahead. In contrast, tags are generated from the PC of the prediction-target block and the history at the beginning of that block. By applying ahead pipelining only to index generation and performing non-ahead matching using tags that include the target-block PC, MORSL avoids extra candidate reads caused by missing history. The only additional hardware required by this organization is one stage of pipeline registers that hold the ahead-read results. The final TAGE prediction is selected by a meta-predictor from the prediction based on the longest match prediction and HCPred, which is based on the longest match among high-confidence entries [21].

B. Tagged corrector (TC)

The tagged corrector (TC) is an override-style auxiliary predictor that replaces a summation-based SC. The TC exploits

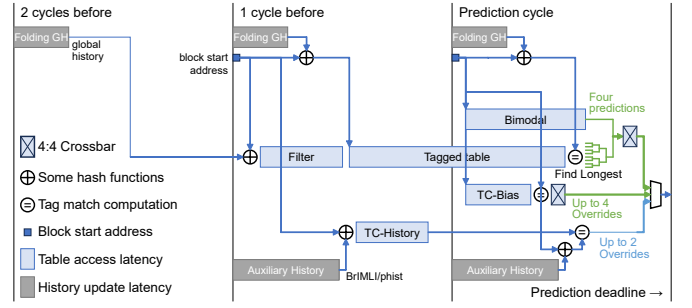


Fig. 2. Timing Diagram of MORSL.

auxiliary correlations through selective overrides by high-confidence tagged entries rather than through the sum of multiple counter outputs. The TC mainly consists of TC-bias, which handles statistical biases in branch directions, and TC-history, which handles correlations based on history types different from those used by TAGE.

1) *TC-bias*: TC-bias is a tagged corrector indexed only by the start PC of the prediction block. TC-bias handles cases where branch directions exhibit statistical bias, but TAGE has difficulty finding effective long-history correlations. Each TC-bias entry has a 7-bit tag that identifies a block, as well as 3-bit direction prediction counters and 3-bit confidence counters for four ranks. TC-bias allocates an entry when TAGE mispredicts. When the prediction counter is in a strong state ($-4/-3/2/3$) and the confidence counter is high ($1/2/3$), TC-bias overrides the TAGE prediction. TC-bias always updates the direction prediction counter on a tag hit. TC-bias updates the confidence counter only when the direction prediction counter is in a strong state.

2) *TC-history*: TC-history is also an override-style tagged branch predictor. TC-history captures correlations based on a history that combines BrIMLI and path history [21]. Each TC-history entry has an 8-bit tag that identifies the context, including two rank bits, a 3-bit prediction counter, and a 4-bit confidence counter. Because TC-history is 2-way set-associative, each SRAM word contains two such entries.

A TC-history entry is allocated when TAGE with TC-bias mispredicts. When the confidence counter is high (2–7), TC-history overrides the TAGE with TC-bias prediction regardless of the strength of the prediction counter. TC-history always updates both the prediction counter and the confidence counter. TC-history generates the index and accesses the tagged table using the PC and history from one block earlier. Finally, it performs tag matching using a tag generated from the PC of the prediction-target block and the history at the beginning of that block.

C. Allocation-Guided Access Filter

1) *Motivation*: In TAGE, it is challenging to determine at prediction time whether long-history tagged banks are unnecessary. Most branches can be predicted accurately using only directional bias or short-history correlations. For such branches, bypassing accesses to long-history tagged banks

can reduce dynamic energy. However, making this decision at prediction time with low overhead is not straightforward. The predictor must determine whether a branch context can be handled by short histories alone, or whether a useful entry may exist in a long-history tagged bank. In other words, before skipping accesses to long-history tagged banks, the predictor must infer, without reading those banks, that they are unlikely to contain a useful matching entry.

To address this problem, we exploit the allocation rule of TAGE. In TAGE, when a misprediction occurs, new entries are allocated in one or more tagged banks whose history lengths are longer than that of the provider. Thus, a past allocation of a long-history entry for a given branch context provides conservative evidence that shorter histories alone were insufficient for that context. Conversely, if no such allocation has been recorded, the corresponding long-history tagged banks are unlikely to contain a useful matching entry.

2) *Implementation:* Based on this observation, we introduce an allocation-guided access filter. The allocation-guided access filter is a Bloom-filter-like mechanism that records, with low overhead, whether a long history entry has been allocated in the past. When the access filter determines that long-history tagged banks are unlikely to be needed, it operates the TAGE component of MORSL in mini-TAGE mode, which reads only the two short-history banks (h6 and h7). The allocation-guided

TABLE I
KEY ORGANIZATION PARAMETERS OF MORSL COMPONENTS

Item	Parameter
Base predictor	4 KiB bimodal; single SRAM for prediction bits; four lane-wise SRAM banks for hysteresis bits
Global histories	8 history lengths: 5, 7, 11, 17, 33, 65, 126, and 334 blocks; 6-bit insertion; 3-bit shift for short histories and 1-bit shift for long histories
Tagged components	8 logical components; 8 physical banks, 2048 entries per bank, and 2:2 interleaving
Tagged entry	12-bit tag, 3-bit prediction counter, and 1-bit usefulness; the upper 2 tag bits encode the lane
Allocation-guided access filter	2 arrays, one is indexed only by the block PC, the other is indexed as gshare; 2048 entries each
TC-bias	128 entries, 2 banks
TC-history	128 sets, 2-way, 2 banks

TABLE II
ABLATION STUDY OF MORSL AND COMPARISON WITH O-GEHL

Branch Predictor	MPKI	EPI _{cbp}	Latency	VFS
Bloom + O-GEHL (no ahead)	5.085	462	598	
+ BrIMLI/phist	5.057	481	600	
Bloom + O-GEHL (1-block ahead)	5.106	471	300	0.9911
+ BrIMLI/phist	5.066	490	300	0.9913
TAGE (no ahead)	4.954	491	495	
+ bank interleave	4.876	501	557	
+ AG access filter	4.880	419	569	
+ random reset	4.897	400	569	
+ TC-bias	4.845	440	578	
+ TC-history	4.763	484	580	
TAGE (1-block ahead)	5.006	510	277	0.9914
+ bank interleave	4.923	520	277	0.9920
+ AG access filter	4.923	455	277	0.9929
+ random reset	4.944	433	277	0.9931
+ TC-bias	4.885	480	286	0.9932
+ TC-history (MORSL)	4.803	524	286	0.9935

access filter consists of two 1-bit tables. The first table is indexed only by the PC, and the second table is indexed by the XOR of the PC and the shortest history. The access filter starts reading all tagged banks only when both bits read from the two tables are 1.

The allocation-guided access filter saturates toward a conservative direction. Even if all entries become 1, the TAGE component of MORSL simply behaves like full TAGE, and thus prediction accuracy does not degrade. However, this state eliminates opportunities for energy reduction. Thus, when mini-TAGE correctly predicts all conditional branches in a block, we reset the corresponding filter entries with a very low probability. This reset can introduce false negatives and thereby reduce prediction accuracy, but it restores energy savings by creating opportunities to return to mini-TAGE mode.

V. EXPERIMENTAL RESULTS AND ANALYSIS

We evaluate MORSL using HARCOT, the evaluation library used for CBP-NG. All results are measured on the 168 training traces provided by CBP-NG. Table I summarizes the configuration. The 37.5 KiB configuration of MORSL achieves 4.803 MPKI, an EPI_{cbp} of 524 fJ/ins, an IPC of 8.647, and a VFS of 0.9935. Its critical path latency is 286 ps.

Table II compares prediction accuracy, energy consumption, and latency when each mechanism of MORSL is removed. We do not report VFS for predictors whose prediction latency does not fit within one cycle (300 ps), because the meaning of VFS is unclear in such cases. For reference, the table also includes the results for O-GEHL. Because TAGE uses a single entry as the basis for prediction, it is more sensitive than O-GEHL to the increase in entry conflicts caused by ahead pipelining. Even so, the accuracy loss due to ahead pipelining is small, at about 1%. Bank interleaving and the allocation-guided access filter substantially increase prediction latency, but this latency increase can be hidden by 1-block-ahead pipelining. The energy reduction achieved by the allocation-guided access filter becomes smaller with ahead pipelining, because the access cannot be filtered if any of the ahead candidates cannot be predicted by mini-TAGE.

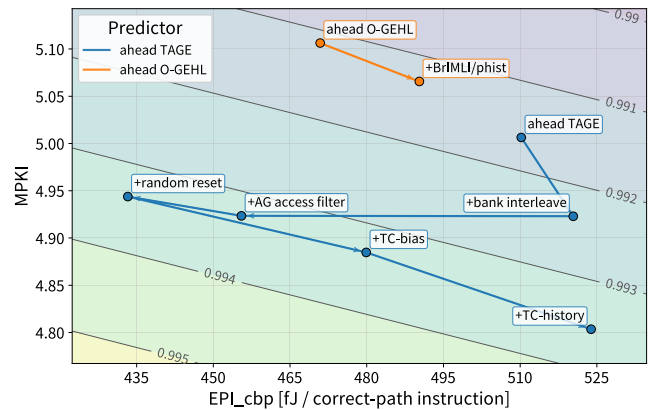


Fig. 3. Comparison of MPKI and EPI_{cbp} for branch predictors.

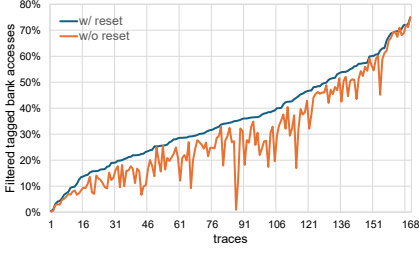


Fig. 4. Filtering rate of the access filter.

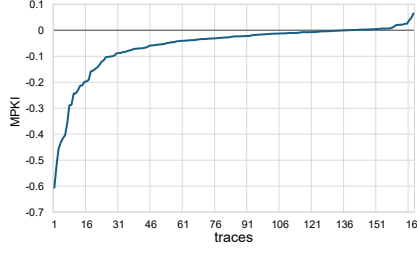


Fig. 5. The TC-bias MPKI reduction S-curve.

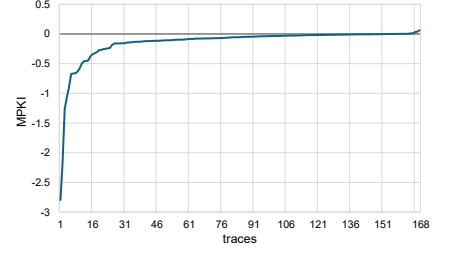


Fig. 6. The TC-history MPKI reduction S-curve.

Figure 3 plots several branch predictors with MPKI on the vertical axis and EPI_{cbp} on the horizontal axis. The VFS contours indicate the maximum VFS that can be achieved for a given pair of MPKI and EPI , and therefore do not necessarily coincide with the actual VFS of each branch predictor. The figure shows that O-GEHL and TAGE achieve almost the same VFS as baseline predictors. Relative to this baseline, bank interleaving provides a large accuracy improvement with only a small increase in energy. The allocation-guided access filter further reduces energy substantially. Applying these two techniques to O-GEHL is difficult, which is why MORSL uses TAGE as its base predictor. Adding TC-bias and TC-history moves the design point along a trade-off between increased energy and improved accuracy, almost parallel to the VFS contours. However, the contribution of the accuracy improvement is slightly larger, resulting in a higher VFS.

Figures 4, 5, and 6 show S-curves of the benefits of the allocation-guided access filter, TC-bias, and TC-history. The allocation-guided access filter suppresses accesses to six of the eight tagged-table banks and can therefore reduce tagged-table accesses by up to 75%. Across the 168 traces, it suppresses 35.9% of tagged-bank accesses on average, reducing the average EPI_{cbp} by 87 fJ/ins with only a 0.021 increase in MPKI. This MPKI increase is caused by the random reset mechanism, and the filter introduces virtually no MPKI degradation without the reset. Without the reset, the filter suppresses 29.7% of tagged-bank accesses on average; the reset greatly recovers filterable opportunities. TC-bias and TC-history improve prediction accuracy with modest energy overheads: TC-bias reduces MPKI by 0.059 on average with an energy overhead of 47 fJ/ins, while TC-history further reduces MPKI by 0.081 over TAGE+TC-bias with an energy overhead of 44 fJ/ins. After accounting for IPC degradation caused by RAM read-modify-write operations, TC-bias and TC-history improve VFS by 0.0002 and 0.0003, respectively.

VI. DISCUSSION AND FUTURE WORK

Because our predictor is based on TAGE, its overall prediction-accuracy characteristics are largely similar to those of TAGE. The gradual saturation of the allocation-guided access filter toward an all-1 state did not cause a problem in the short-trace evaluation of CBP-NG, but it may become an issue in practical use. Addressing this issue is left for future work. Our preliminary study showed that TC-bias and TC-history can provide substantial accuracy improvements when sufficient

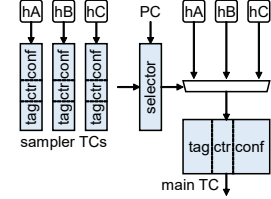


Fig. 7. TC-history with a sampler.

storage capacity is available, whereas under the CBP-NG rules, it was optimal to allocate only a very small storage budget to them.

Our preliminary study showed that TC-history can further reduce mispredictions by dynamically selecting an appropriate history type from multiple candidates. Figure 7 shows such an organization, where three history types, hA, hB, and hC, are used as candidates, and a per-PC selector chooses the history type used by the main TC-history. When the main TC-history mispredicts, a tiny sampler TC-history with only about eight entries tentatively learns another history type; if the sampler prediction proves useful, the selected history type for that PC is recorded in the selector. Although such a small sampler effectively reduced mispredictions in our evaluation, the improvement did not translate into a higher VFS when the additional cost of the selector and sampler was included. Therefore, MORSL does not adopt this mechanism. Low-cost integration of TC-history with multi-history selection in a way that improves VFS remains future work.

VII. CONCLUSION

To bridge the gap between academic high-accuracy branch predictors and branch predictors for commercial processors, we used HARCOT to explore how realistic branch predictors should be organized and proposed MORSL as a result. MORSL consists of a TAGE component built from narrow-word SRAMs, a tagged corrector, and an allocation-guided access filter. MORSL also carefully chooses whether each predictor component should be accessed ahead of the prediction target, reducing the accuracy loss caused by ahead pipelining. We implemented MORSL with these mechanisms in HARCOT and evaluated it with 168 training traces provided by CBP-NG. MORSL achieves a prediction throughput of 8.647 ins/cycle, a prediction accuracy of 4.803 MPKI, and an energy consumption of 524 fJ/ins, resulting in a VFS of 0.9935. Its critical path latency is reported as 286 ps.

REFERENCES

- [1] “Championship Branch Prediction (CBP-1),” <https://jilp.org/cbp/>, 2004, accessed 2026-05-08.
- [2] “Championship Branch Prediction (JWAC-2),” <https://jilp.org/jwac-2/>, 2011, accessed 2026-05-08.
- [3] “Championship Branch Prediction (CBP-4),” <https://jilp.org/cbp2014/>, 2014, accessed 2026-05-08.
- [4] “Championship Branch Prediction (CBP-5),” <https://jilp.org/cbp2016/>, 2016, accessed 2026-05-08.
- [5] “6th Championship Branch Prediction (CBP2025),” <https://ericrotenberg.wordpress.ncsu.edu/cbp2025/>, 2025, accessed 2026-05-08.
- [6] “The simulator for the Next-Generation Championship in Branch Prediction (CBP-NG),” <https://github.com/AmpereComputing/cbp-ng>, 2026, accessed 2026-05-08.
- [7] N. Adiga, J. Bonanno, A. Collura, M. Heizmann, B. R. Prasky, and A. Saporito, “The IBM z15 High Frequency Mainframe Branch Predictor Industrial Product,” in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 27–39.
- [8] L. Cai, A. Deshmukh, and Y. Patt, “Enabling Ahead Prediction with Practical Energy Constraints,” in *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, ser. ISCA ’25. New York, NY, USA: Association for Computing Machinery, 2025, pp. 559–571. [Online]. Available: <https://doi.org/10.1145/3695053.3730998>
- [9] B. Grayson, J. Rupley, G. Zuraski, E. Quinell, D. A. Jiménez, T. Nakra, P. Kitchin, R. Hensley, E. Brekelbaum, V. Sinha, and A. Ghiya, “Evolution of the Samsung Exynos CPU Microarchitecture,” in *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture*, ser. ISCA ’20. IEEE Press, 2020, pp. 40–51. [Online]. Available: <https://doi.org/10.1109/ISCA45697.2020.00015>
- [10] Y. Ishii, J. Lee, K. Nathella, and D. Sunwoo, “Re-establishing Fetch-Directed Instruction Prefetching: An Industry Perspective,” in *Proceedings of the 2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2021, pp. 172–182.
- [11] D. A. Jiménez, “Multiperspective Perceptron Predictor,” in *Proceedings of the 5th Championship Branch Prediction Workshop (CBP-5)*, 2016, pp. 1–5.
- [12] —, “Multiperspective Perceptron Predictor with TAGE,” in *Proceedings of the 5th Championship Branch Prediction Workshop (CBP-5)*, 2016, pp. 1–5.
- [13] —, “Multiperspective Perceptron Predictor,” in *Proceedings of the 6th Championship Branch Prediction Workshop (CBP2025)*, 2025, pp. 1–6.
- [14] T. Koizumi, T. Maekawa, M. Mizuno, M. Kuroki, T. Tsumura, and R. Shioya, “RUNLTS: Register-value-aware Predictor Utilizing Nested Large Tables,” in *Proceedings of the 6th Championship Branch Prediction Workshop (CBP2025)*, 2025, pp. 1–6.
- [15] G. H. Loh, D. S. Henry, and A. Krishnamurthy, “Exploiting Bias in the Hysteresis Bit of 2-bit Saturating Counters in Branch Predictors,” *Journal of Instruction-level Parallelism*, vol. 5, pp. 1–32, 2003.
- [16] P. Michaud, A. Seznec, and R. Uhlig, “Trading conflict and capacity aliasing in conditional branch predictors,” in *Proceedings of the 24th Annual International Symposium on Computer Architecture*, ser. ISCA ’97, 1997, pp. 292–303.
- [17] D. Sanchez, L. Yen, M. D. Hill, and K. Sankaralingam, “Implementing Signatures for Transactional Memory,” in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO 40. USA: IEEE Computer Society, 2007, pp. 123–133.
- [18] A. Seznec, “A 64 Kbytes ISL-TAGE branch predictor,” in *Proceedings of the 3rd Championship Branch Prediction Workshop (CBP-3)*, 2011, pp. 1–4.
- [19] —, “TAGE-SC-L Branch Predictors,” in *Proceedings of the 4th Championship Branch Prediction Workshop (CBP-4)*, 2014, pp. 1–8.
- [20] —, “TAGE-SC-L Branch Predictors Again,” in *Proceedings of the 5th Championship Branch Prediction Workshop (CBP-5)*, 2016, pp. 1–4.
- [21] —, “TAGE: an engineering cookbook,” HAL Inria, RR-9561, 2024, submitted on December 4, 2024; Last modified March 28, 2025. [Online]. Available: <https://hal.science/hal-04804900>
- [22] A. Seznec, S. Felix, V. Krishnan, and Y. Sazeides, “Design Tradeoffs for the Alpha EV8 Conditional Branch Predictor,” in *Proceedings of the 29th Annual International Symposium on Computer Architecture*, ser. ISCA ’02, 2002, pp. 295–306.
- [23] A. Seznec and P. Michaud, “A case for (partially) TAgged GEometric history length branch prediction,” *Journal of Instruction-level Parallelism*, vol. 8, pp. 1–23, 2006.

APPENDIX — THE HARCOM IMPLEMENTATION DETAILS

SRAM BUDGET BREAKDOWN

Table III shows the SRAM budget breakdown of 37.5 KiB MORSL, and Figure 8 shows its floorplan. MORSL can be built with 51 single-port SRAMs, which allow only one read or one write per cycle and are the only SRAMs that can be used in the HARCOM library.

A. Base predictor — 5 SRAMs, 4 KiB, 1474 μm^2

The base predictor is built with 5 SRAMs. The bimodal prediction array is built with a monolithic SRAM to reduce access energy. The bimodal hysteresis array requires only one write on a correct prediction, because it adopts the split-counter form [15], [22], and should support partial writes. Considering HARCOM constraints, we built it using 4 SRAMs.

B. Tagged tables — 40 SRAMs, 32 KiB, 10074 μm^2

Each tagged component entry in TAGE contains a (partial) tag, a 3-bit counter, and a useful bit. The 3-bit counter consists of a prediction bit (the uppermost bit) and two hysteresis bits. We classify these fields into two groups based on whether they change with a correct prediction: unchanged ones (the tag and prediction bits) and changeable ones (the hysteresis bits and the useful bit). We manage each of these groups differently.

The former is stored in 8 SRAMs, each corresponding to a physical bank. The tag and the prediction bit never change on a correct prediction; thus, we merge them into 1 SRAM per bank. There are some situations where the direction changes while the tag remains unchanged, but they occur rarely. In such situations, the whole entry is rewritten. It may consume large amounts of energy, but since they occur rarely, it does not offset the energy saved by merging SRAMs.

The latter is stored in 32 SRAMs, for 4 banks per 8 physical banks. The hysteresis bits and the useful bit need to be updated even when a prediction is correct. This necessitates a dual-port memory, but a 4-bank structure practically resolves this problem [23]. We merge the hysteresis bits array and the useful bit array, but this degrades prediction accuracy because it resets the hysteresis bits whenever the useful bit is reset, since HARCOM’s SRAM supports only full resets. We confirmed that it does not offset the energy reduction from merging SRAMs and that it is beneficial for VFS. However, in a realistic implementation, we should not use simultaneous resetting.

C. Allocation-Guided Access Filter — 2 SRAMs, 0.5 KiB, 430 μm^2

The allocation-guided access filter is built using 2 SRAMs, each corresponding to a hash function in the egskew-like [16] parallel Bloom [17] filter. The allocation-guided access filter needs to be written only when a new tagged entry is allocated; thus, it can be built with single-port SRAMs. Note that the random resetting feature requires write ports. In such a situation, we stall one cycle (request an extra cycle). Since the random reset occurs infrequently, it has little effect on the overall IPC.

D. TC-bias and TC-history — 2 SRAMs each, <0.5 KiB each, 706 μm^2 total

The TC-bias and TC-history are each built using 2 SRAMs, for a total of 4 SRAMs. The TC-bias and TC-history need to be updated even on a correct prediction. We confirmed that the banking solution used in the tagged table degrades prediction accuracy due to a stale counter, which might be a HARCOM-specific problem. To address this problem, we used (1) entry forwarding, (2) guaranteed write, and (3) a 2-bank structure. Entry forwarding resolves the stale counter problem, but it might indicate that the speculative counter update [13] is needed. We guarantee that any write is committed by stalling appropriately. It degrades the overall IPC, but using a 2-bank structure greatly mitigates this. The TC-bias and TC-history degrade the overall IPC by about 0.1 each, but they hardly affect the VFS.

LATENCY OF MAJOR CRITICAL PATHS

MORSL consists of a few components and has individual critical-path timings. The following are major critical path timings. These critical paths could be further optimized, but the required cycle time is 300 ps, so we leave them unoptimized.

A. Base predictor — 281 ps

The bimodal predictor provides base predictions of TAGE. It should not be ahead-pipelined to achieve high prediction accuracy. A 2 KiB table read requires 166 ps. The total prediction latency is 281 ps, including 9 ps for the 2:1 MUX by the meta predictor, 19 ps for the 4:4 crossbar, 9 ps for the TC-overriding 3:1 MUX, and 18 ps for the final FO64.

B. Access filter and Tagged tables — 580 ps

The access filters and the tagged tables are accessed 1-block ahead. These accesses need to be serialized. Each 1-bit read from a 2048-entry Bloom table requires 109 ps. The filtering decision is generated in 158 ps. Each 13-bit read from a 2048-entry TAGE’s gtagpred table requires 172 ps. A round trip to the farthest gtagpred table takes 47 ps each way. The total prediction latency is 580 ps, including the same circuits as the base predictor. Since they are 1-block ahead-pipelined and the cycle time is 300 ps, the effective latency is 280 ps.

C. TC-bias — 258 ps

TC-bias overrides TAGE prediction and should not be ahead-pipelined to achieve high prediction accuracy. Each 31-bit read from a 1984-bit table requires 76 ps. The total prediction latency is 258 ps, including 19 ps for the 4:4 crossbar, 9 ps for the TC-overriding 3:1 MUX, and 18 ps for the final FO64.

D. TC-history — ~ 350 ps

TC-history overrides TAGE+TC-bias prediction and can be ahead-pipelined. Each 30-bit read from a 1920-bit table requires 76 ps. The total prediction latency appears to be about 350 ps due to the long latency of the BrIMLI computation, including backwardness decision. Since it is ahead-pipelined, the

table read is not on the critical path. The table-read bypassing constitutes a critical path with 261 ps latency, including the same circuits as TC-bias.

E. Meta predictor update — 286 ps

The meta predictor selects between the longest-match prediction and the not-weak longest-match prediction. It is a 4-bit counter and is shared by the lanes. The update requires 4-bit saturating addition, which incurs long latency; it is on the MORSL critical path (286 ps). Note that this update can be pipelined [6].

TABLE III
SRAM BUDGET BREAKDOWN OF 37.5 KiB MORSL.

Component	Entry structure	# of entries	Storage (bits)	# of SRAMs	Port usage			Note
					Predict	Correct pred.	Mispred.	
Bimodal prediction array	prediction (4×1)	2^{12}	16384	1	1R	-	1W	
Bimodal hysteresis array	hysteresis (1)	4×2^{12}	16384	4	-	1W	1R + 1W	4 banks for partial write
Tagged table (gtagpred)	tag (12), prediction (1)	8×2^{11}	212992	8	1R	-	1W	
Tagged table (uhyst)	useful (1), hysteresis (2)	8×2^{11}	49152	32	1R	1W (4 banks)	1W	4 banks for dual port*
Access filter	long history allocated (1)	2×2^{11}	4096	2	1R	-	1W	2 different hash functions
TC-bias	tag (7), ctr (4×3), conf. (4×3)	2^7	3968	2	1R	1W (2 banks)	1W	2 banks for dual port
TC-history	$2 \times \{\text{tag (8), ctr (3), conf. (4)}\}$	2^7	3840	2	1R	1W (2 banks)	1W	$\uparrow$, 2-way set associative
Total			306816	51				

*: The uhyst table needs to be updated even if the prediction is correct; using a 4-bank structure is a cost-effective alternative to the use of dual-ported tables [23].

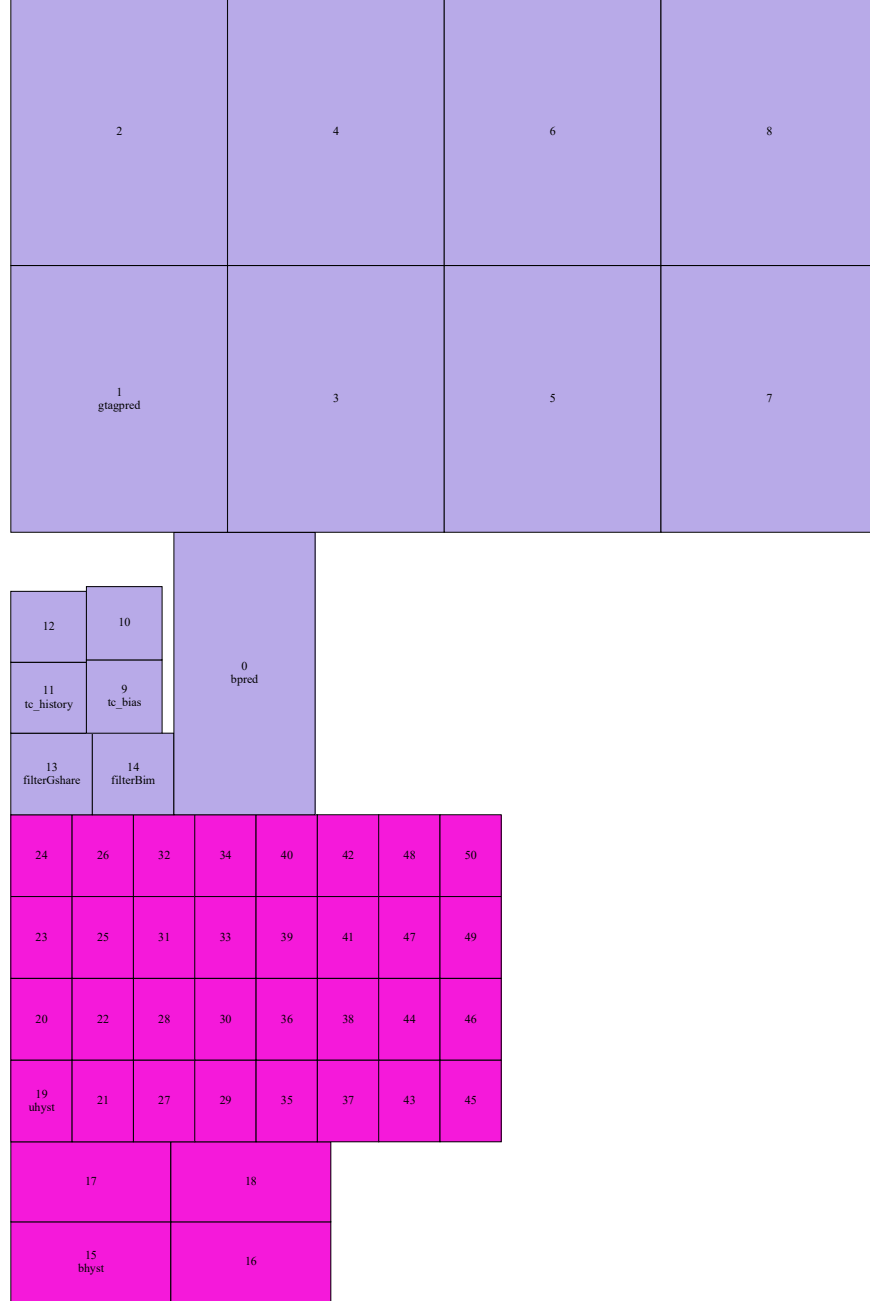


Fig. 8. 37.5 KiB MORSL floorplan.